УДК 004.056 .32

Вестник УрФО № 2(32) / 2019, с. 5–11

Белов Д. П., Зулькарнеев И. Р., Крамаренко Р. В., Пелевин И. А.

DOI: 10.14529/secur190201

РЕАЛИЗАЦИЯ ОТКАЗОУСТОЙЧИВОСТИ В СИСТЕМЕ ОРКЕСТРАЦИИ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ KUBERNETES

В данной статье авторами описываются результаты эксперимента по реализации разработанной методики обеспечения отказоустойчивости микросервисных архитектур. Описываются критерии оценки методики, структура тестовой системы на базе контейнеров Docker в системе оркестрации Kubernetes. Дается описание настроек и рекомендации для всех подсистем в разрезе обеспечения отказоустойчивости. Выполняется подбор программных продуктов для реализации тестовой отказоустойчивой системы на основе микросервисной архитектуры. Приводятся результаты апробации методики и варианты дальнейшего её усовершенствования.

Ключевые слова: *Микросервисная архитектура, Docker, Kubernetes, отказоустойчивость..*

Belov D. P., Zulkarneev I. R., Kramarenko R. V., Pelevin I. A.

PROVIDING FAULT TOLERANCE FOR MICROSERVICE ARCHITECTURES IN KUBERNETES, THE CONTAINER ORCHESTRATION ENGINE

The following article represents the results of the experiment for the method of providing fault tolerance for microservice architectures. Authors suggest method criteria, describe the structure of the test Docker-containerized system in Kubernetes, the container orchestration engine. In the article, there are recommendations and description of all subsystem's fault tolerance providing settings. Also, we choose software for fault tolerance microservice cluster realization. The results of the method approbation are represented and future improvements are described.

Keywords: *Microservice architecture, Docker, Kubrenetes, fault tolerance.*

При реализации сложных разноплановых ИТ-систем все чаще предпочтение отдают виртуализации с использованием микросервисной архитектуры. По данным DZone 53% предприятий используют микросервисы для разработки или продакшена[1]. В статье "Методика построения отказоустойчивых систем на базе микросервисной архитектуры" авторами было выявлено, что одной из основных причин возникновения трудностей при работе с подобными архитектурами, является отсутствие единых формализованных требований по обеспечению отказоустойчивости[2].

В связи с этим авторами была разработана Методика построения отказоустойчивых систем на базе микросервисной архитектуры. Ее реализация возможна несколькими способами в зависимости от характеристик информационной системы, количества и квалификации персонала и внутренних целей и задач.

Согласно методике для бесперебойного и оптимального функционирования система должна удовлетворять следующим требованиям:

1) Репликация.

2) Снижение времени на служебные операции.

3) Централизация управления ресурсами.

4) Мониторинг состояния компонентов.

5) Снижение скорости восстановления при сбоях.

6) Автоматизация работ по управлению системой, уменьшение роли человека при настройке и восстановлении работоспособности[2].

В качестве критериев оценки реализации и эффективности методики были выбраны:

1) степень автоматизации работ по управлению системой;

2) время восстановления системы;

3) время восстановления работоспособности сервиса;

4) время восстановления работоспособности группы сервисов;

5) время восстановления при отказе сервера;

6) время уведомления об инцидентах.

Для удовлетворения указанным выше требованиям система должна состоять из: подсистемы оркестрации микросервисов, подсистемы сбора логов, подсистемы мониторинга состояния, подсистемы хранения данных, ядра.

В качестве подсистемы оркестрации сервисов был выбран фреймворк Kubernetes. Kubernetes не привязан к аппаратной инфраструктуре и представляет весь центр хранения и обработки данных как единый вычислительный ресурс. Он позволяет разворачивать и запускать программное обеспечение без необходимости знания подробностей работы сервера. При разворачивании многокомпонентного приложения Kubernetes самостоятельно выбирает сервер для каждого компонента, устанавливает компонент и обеспечивает его видимость и связь с другими компонентами приложения [3]. Одним из основных понятий Kubernetes является нода — физический узел системы, на котором расположены различные элементы. Для тестовой системы была выбрана реализация данной системы, подразумевающая три мастер-ноды, три основных ноды и три etcd сервера, с возможностью расширения.

На мастер-нодах расположены системные сервисы, необходимые для работы Kubernetes. Они следят за состоянием всех компонентов, расположенных на основных нодах.

На основных нодах находится основная часть системы — контейнеры (образ и запускаемая в нем команда) с микросервисами. Совокупность контейнеров с общими характеристиками (IP-адресом, сеть, общие хранилища данных, лейблы и т.д.) в Kubernetes именуются подами. В случае если под перестает отвечать на запросы системных сервисов, расположенных на мастер-ноде, он перезапускается. В случае, если перестает отвечать вся нода, все сервисы расположенные на ней в течении минуты перезапускаются на другой ноде.

В ходе проведения экспериментов с функционированием тестовой системы было определено, что для корректной работы кластера необходимо наличие как минимум двух функционирующих etcd-серверов. Они представляют собой распределенное хранилище данных типа "ключ-значение" и используются в качестве базы данных для Kubernetes.

Для обеспечения мониторинга состояния составных элементов системы и снижения затрат человеко-часов на поиск и устранение неисправностей были реализованы тестовые системы мониторинга и централизованного сбора логов.

Система мониторинга организована при помощи подсистемы Grafana для графического отображения и InfluxDB в качестве хранилища данных мониторинга.

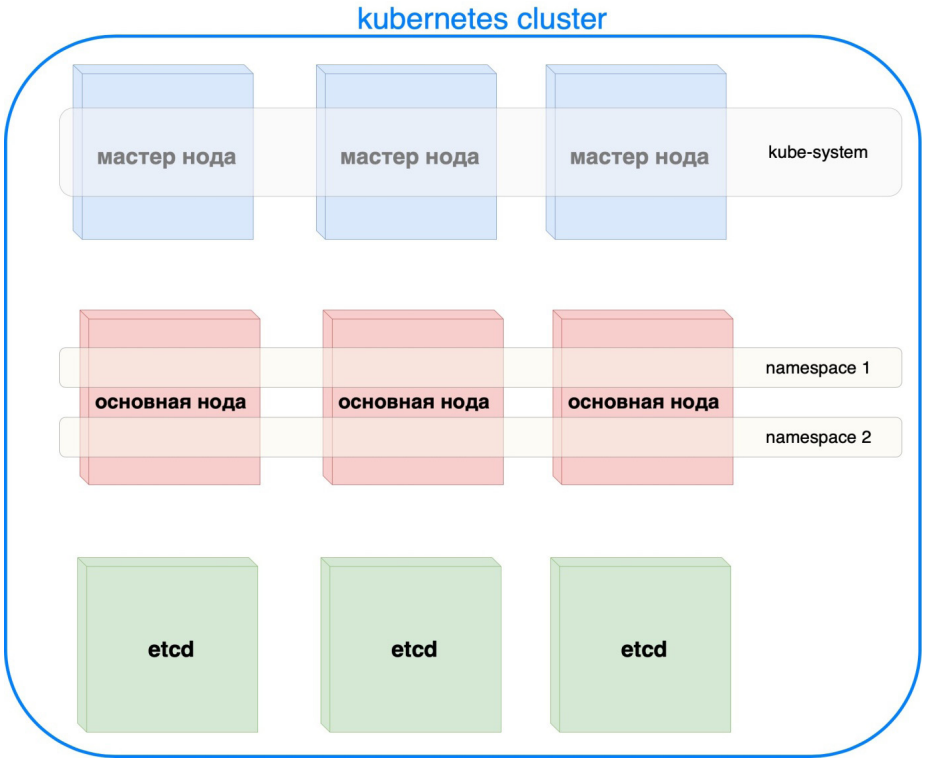

Рис.1. Тестовая реализация отказоустойчивого кластера

InfluxDB является бессхемной базой данных для хранения временных записей с открытым исходным кодом с опциональными компонентами с закрытым исходным кодом разработанная InfluxData. В настоящий момент InfluxDB для хранения использует встроенную структуру данных: временное структурированное дерево слияния (TSM-дерево). Согласно исследованиям сотрудников Брюссельского открытого университета данный формат не подвержен проблемам удаления, что обеспечивает в 45 раз более эффективное сжатие на диске по сравнению с B+-деревом: [4].

Grafana позволяет запрашивать, визуализировать отправлять предупреждения и принимает метрику вне зависимости от того, где она хранится. Данная подсистема дает возможность оптимально организовать управление ресурсами, пользователями и осуществление настройки в больших распределенных системах, объединяющих несколько крупных подразделений или отдельных организаций.

В качестве сервиса для сбора метрик, было выбрано готовое решение Heapster, которое собирает и интерпретирует различные данные, такие как использование вычислительных ресурсов, события жизненного цикла и т. д. Данное решение интегрируется в Kubernetes, собирая помимо системной информации, данные о функционировании сервисов в Kubernetes. Все собранные метрики он отправляет в InfluxDB. По хранящимся в InfluxDB метрикам можно строить различные графики в Grafana, в которых мы задаем критичные значения для уведомления администратора. Уведомление возможно различными способами, в том числе через мессенджер, что позволяет обеспечивать быструю реакцию на возникший инцидент безопасности.

Для системы централизованного логирования в тестовой среде были выбраны следующие подсистемы: Kibana для отображения поступающих событий и Elasticsearch в качестве хранилища записей логирования, а также FluentD, как агрегатор логов.

Подсистема Elasticsearch осуществляет полнотекстовый поиск и аналитику записей логирования в условиях постоянного использования распределенных вычислений и распределенного хранения данных.

В работе Бхарви Диксита отмечено, что Elasticsearch основан на REST-архитектуре и позволяет выполнять не только CRUD операции через HTTP, но выполнять мониторинг кластера с использованием REST API[5]. Он сконструирован для горизонтального, а не

вертикального расширения. Построение системы можно начать с кластера Elasticsearch из одной ноды на ноутбуке и превратить в сотни тысяч нод не беспокоясь о сложностях системы, связанных с распределенными вычислениями, распределенным хранением документов и поиском.

Подсистема Kibana является веб-приложением, которое представляет данные, обрабатываемые Elasticsearch. Она не загружает данные из Elasticsearch для из дальнейшей обработки, но использует мощности Elasticsearch для выполнения всех ресурсоемких задач. Все это обеспечивает отображение информации в реальном времени. С ростом количества данных кластер Elasticsearch масштабируется относительно этого количества для обеспечения минимальной задержки, согласно SLA[6].

Первым шагом в настройке системы логирования, был запуск и конфигурирование сервиса Fluentd. Он настраивается для сбора логов всех контейнеров запущенных на нодах и отправки их в Elasticsearch.

Источник для сбора логов — папка "/var/log/containers/", в которую пишут логи все микросервисы. В конфигурационном файле fluentd это конфигурируется в подобном формате:

1) указывается плагин для чтения файлов (tail);

2) настраивается путь к лог файлам;

3) задается файл для записи позиции последнего чтения;

4) указывается формат даты времени;

5) задается тэг для выбираемых логов (Kubernetes.*);

6) выбирается необходимый формат логов (json).

Собранные логи отправляются по указанным хосту и порту, с префиксом Kubernetes-<имя сервиса>, в конфигурации это отражено следующим образом:

1) указывается какой плагин использовать для ввода (elasticsearch_dynamic);

2) задаются хост и порт;

3) настройка хранения поля тэга;

4) fluentd настраивается с использованием стандартного формата именования индекса;

5) задается желаемый префикс для индекса;

6) настраивается количество потоков.

Далее запускается Elasticsearch, в нем ежедневно автоматически создаются индексы (хранилища), в которые приходят все логи собранные fluentd, а также отправляемые из микросервисов вручную. Данные логи можно фильтровать по индексам или каким либо параметрам в Kibana и просматривать в удобном виде.

Для удовлетворения требования к репликации данных был реализован кластер баз данных (набор баз, управляемых одним экземпляром работающего сервера) на серверах PostgreSQL типа мастер-слейв в следующем виде: несколько виртуальных машин, одна из из которых считается активной, а остальные — пассивно копируют данные для замены основной в случае выхода из строя.

Экспериментальный прототип кластера был реализован следующим образом:

1. Установка программного обеспечения баз данных postgresql, менеджера репликаций repmgr, программного обеспечения для управления пулом соединений: pgbouncer.

2. Настройка ssh соединения без использования пароля для аутентификации между всеми нодами и к серверу через пользователя postgres.

3. Настройка окружения (переменной Linux PATH для определения пути к исполняемому файлу postgres, переменной PGDATA для указания пути к файлам конфигурации и данных postgresql).

4. Для мастера:

– указание параметра для принятия запросов от необходимых ip-адресов;

– указание уровня логирования hot_standby; таким образом выполняется логирование достаточного количества информации для восстановления транзакций, при этом обеспечивая достаточную производительность кластера и отсутствие избытков информации;

– включение архивирования для передачи полных сегментов журнала предзаписи (WAL) в хранилище;

– отключение хранения сегментов WAL (задание значения 0 в конфигурационном файле);

– задание необходимого количества слотов репликации, поддерживаемых сервером (для тестовой системы было выбрано 3 слота);

– разрешение подключения к серверу для отправки SQL-запросов в процессе восстановления;

– настройка максимального количества процессов передачи WAL работающих одновременно и максимального количества соединений;

– задание команды для архивации завершенного сегмента WAL, порта и дополнительных параметров;

– настройка доверенных сетей для передачи данных в файле pg_hba.conf;

– создание роли repmgr с правами суперпользователя и отсутствием наследования прав (т.е. роль не будет “наследовать” права ролей, членом которых она является);

– разрешение repmgr выполнять вход на сервер (роль может стать начальным авторизованным именем при подключении клиента);

– создание базы данных для repmgr, выдача данному пользователю всех разрешений на созданную базу;

– настройка имени кластера;

– настройка номера, имени ноды и используемого слота репликации для мастера;

– настройка информации для соединения и указание repmgr пути к исполняемому файлу postgresql;

– регистрация сервера как master.

5. Для слейва:

– настройка имени кластера;

– настройка номера, имени ноды и используемого слота репликации для мастера;

– настройка информации для соединения и указание repmgr пути к исполняемому файлу postgresql;

– регистрация standby сервера и регистрация в кластере.

Централизация управления ресурсами осуществлялась при помощи создания “ядра” системы. Ядро представляло собой отдельный сервис обрабатывающий запросы от остальных и отвечающий за их бесперебойную коммуникацию. Экспериментальная реализация ядра была выполнена на языке программирования Python. В соответствии с задачами были реализованы следующие модули: сбор информации о присутствующих в системе сервисах, прием, обработка и отправка сообщений, управление информационной нагрузкой на микросервисы, взаимодействие с фреймворком Kubernetes для создания дополнительных экземпляров контейнеров сервисов. В зависимости от нагрузки в системе может присутствовать одно или несколько ядер.

Таблица 1

Результаты апробации методики

Показатель	До	После
Степень автоматизации работ по управлению системой	Не автоматизированы	Автоматизированы
Время восстановления системы	~5-10 мин	~2 мин
Восстановление работоспособности сервиса	~3-7 мин	~0.5 – 1.5 мин
Время восстановления работоспособности группы сервисов	(3+0.4 * кол-во сервисов) мин	~0.5 – 1.5 мин
Время восстановления при отказе сервера	~4-6 мин	~1-2 мин
Время уведомления о инцидентах	-	~15 - 20 сек

Экспериментальная реализация системы показала эффективность методики для настройки систем, использующих микросер-

висную архитектуру. Полученная система демонстрировала приемлемую отказоустойчивость выполняя автоматическое восстановление сервиса за 40-60 секунд и всей архитектуры за 1.5 минуты, что в 3-7 раз быстрее по сравнению с изначальной системой до применения Методики. Подсистемы логирования и мониторинга своевременно уведомляли об инцидентах и позволяли в случае необходимости выявить и устранить причину неисправности. Своевременные уведомления позволили сократить время выявления неисправностей до 15-20 секунд (вместо неопределенного фактического обнаружения клиентом или сотрудником). Экспериментальное применение Методики было выполнено для архитектур на 10, 20 и 30 микросервисов.

Методика продемонстрировала существенное снижение затрачиваемого на обслуживание системы времени и человеческих ресурсов, а также применимость к любым микросервисным архитектурам вне зависимости от масштаба, и количества сервисов. Система уведомлений своевременно сообщала о неисправностях и необходимости вмешательства квалифицированного сотрудника. Выполнены все требования для определения системы как отказоустойчивой. В дальнейшем планируется ее улучшение и адаптация с использованием других систем оркестрации.

Литература

1. Lawrence Hecht. This Week in Numbers: DevOps Favor Microservices [Электронный ресурс]: «The New Stack» // URL: https://thenewstack.io/week-numbers-devops-favor-microservices/ (дата обращения: 20.09.2018).

2. Белов Д.П., Зулькарнеев И.Р., Крамаренко Р.В. Методика построения отказоустойчивых систем на базе микросервисной архитектуры // Безопасность информационного пространства : сб. тр. XVII Всерос. науч.-практ. конф. студентов, аспирантов и молодых ученых, Челябинск, 29–30 нояб. 2018 г. / отв. за вып. Е. А. Сбродова, М. А. Загребин : в 2 т. Т. 2. Челябинск : Изд-во Челяб. гос. ун-та, 2018. 18-22 с.

3. Marko Lukša, Elesha Hyde, Aleksandar Dragosavljevic, Jeanne Boyarsky, Kevin Sullivan (Eds.). Kubernetes in Action. Manning Publications Co: 2018. NY: 325p.

4. Syeda Noor Zehra Naqvi, Sofia Yfantidou. Time Series Databases and InfluxDB. Universite libre de Bruxelles: 2017. p. 12

5. Bharvi Dixit, Rafał Kuć, Marek Rogoziński, Saurabh Chhajed, Mayur Pawanikar (Ed.). Elasticsearch: A Complete Guide. Packt Publishing Ltd. Birmingham: 2017. 881p.

6. Bahaaldine Azarmi, Safis Editing, Amey Varangaonkar, Prachi Bisht, Dharmendra Yadav. Learning Kibana 5.0. Packt Publishing. Birmingham: 2017. 239p

References

1. Lawrence Hecht. This Week in Numbers: DevOps Favor Microservices [Электронный ресурс]: «The New Stack» // URL: https://thenewstack.io/week-numbers-devops-favor-microservices/ (дата обращения: 20.09.2018).

2. Belov D.P., Zulkarneev I.R., Kramarenko R.V. The methodology of fault tolerance microservices systems. Bezopasnost informacionnogo prostranstva, T2? 2018, 18-22p.

3. Marko Lukša, Elesha Hyde, Aleksandar Dragosavljevic, Jeanne Boyarsky, Kevin Sullivan (Eds.). Kubernetes in Action. Manning Publications Co: 2018. NY: 325p.

4. Syeda Noor Zehra Naqvi, Sofia Yfantidou. Time Series Databases and InfluxDB. Universite libre de Bruxelles: 2017. p. 12

5. Bharvi Dixit, Rafał Kuć, Marek Rogoziński, Saurabh Chhajed, Mayur Pawanikar (Ed.). Elasticsearch: A Complete Guide. Packt Publishing Ltd. Birmingham: 2017. 881p.

6. Bahaaldine Azarmi, Safis Editing, Amey Varangaonkar, Prachi Bisht, Dharmendra Yadav. Learning Kibana 5.0. Packt Publishing. Birmingham: 2017. 239p.

ЗУЛЬКАРНЕЕВ Искандер Рашитович, старший преподаватель, кафедра информационной безопасности, Тюменский государственный университет. 25003, г. Тюменьул. Володарского, д. 6. E-mail: i.r.zulkarneev@utmn.ru

БЕЛОВ Дмитрий Павлович, программист, ООО «Е-софт». 625000, г. Тюмень, ул. 30 лет Победы, д. 81а/1. E-mail: dimonbelov95@gmail.com

КРАМАРЕНКО Роман Владимирович, специалист отдела серверного обеспечения, ООО «Е-софт». 625000, г. Тюмень, ул. 30 лет Победы, д. 81а/1. E-mail: kramromn@gmail.com

ПЕЛЕВИН Игорь Андреевич, аспирант, кафедра информационной безопасности, Тюменский государственный университет, 25003, г. Тюмень ул. Володарского, д. 6. E-mail: i.a.pelevin@utmn.ru

ZULKARNEEV Iskander, Senior Lecturer, Information Security Department, Institute of Mathematics and Computer Science, University of Tyumen. 6 Volodarskogo St., 625003 Tyumen, Russia. E-mail: i.r.zulkarneev@utmn.ru

BELOV Dmitriy, programmer, «E-soft» LTD. 81a/1 30 let Pobedy St., 625000, Tyumen, Russia. E-mail: dimonbelov95@gmail.com

KRAMARENKO Roman, Server interfaces specialist, «E-soft» LTD. 81a/1 30 let Pobedy St., 625000, Tyumen, Russia. E-mail: kramromn@gmail.com

PELEVIN Igor, graduate student, Information Security Department, Institute of Mathematics and Computer Science, University of Tyumen. 6 Volodarskogo St., 625003 Tyumen, Russia. E-mail: i.a.pelevin@utmn.ru